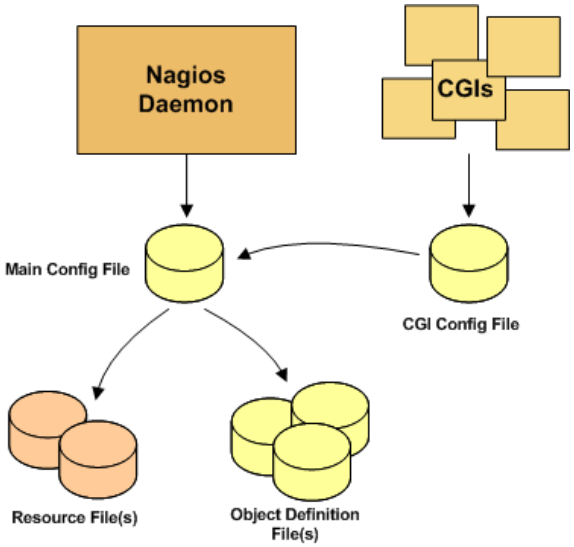
Configure Nagios Core on RHEL/CentOS 8.x

Nagios Configuration:



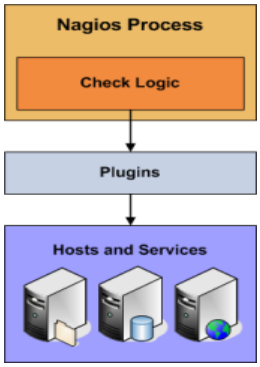
In the previous day, we have seen the installation of Nagios. Introduction

There are several different configuration files that you're going to need to create or edit before you start monitoring anything. Be patient, configuring Nagios Core effectively can take quite a while, especially if you're first-time user. Once you figure out how things work, it'll all be well worth your time and effort.

The configuration files of Nagios are located in /usr/local/nagios/etc

Some Definitions:

Active Checks:



Nagios Core is capable of monitoring hosts and services in two ways: actively and passively. Active checks are the most common method for monitoring hosts and services. The main features of actives check as follows:

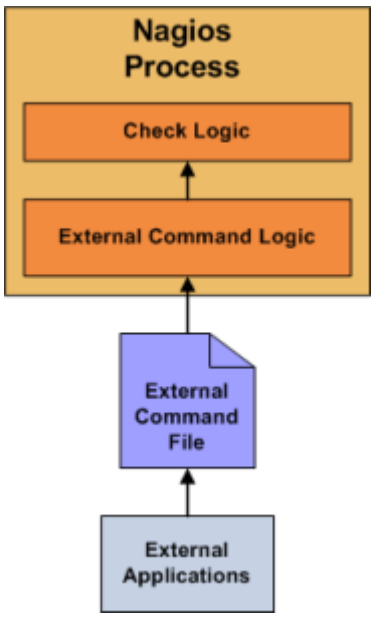
- Active checks are initiated by the Nagios Core process
- Active checks are run on a regularly scheduled basis

When Are Active Checks Executed?

Active checks are executed:

- At regular intervals, as defined by the check_interval and retry_interval options in your host and service definitions
- On-demand as needed

Passive Checks:



In most cases use Nagios Core to monitor your hosts and services using regularly scheduled active checks. Active checks can be used to "poll" a device or service for status information every so often. Nagios Core also supports a way to monitor hosts and services passively instead of actively. The key features of passive checks are as follows:

- Passive checks are initiated and performed by external applications/processes
- Passive check results are submitted to Nagios Core for processing

The major difference between active and passive checks is that active checks are initiated and performed by Nagios Core, while passive checks are performed by external applications.

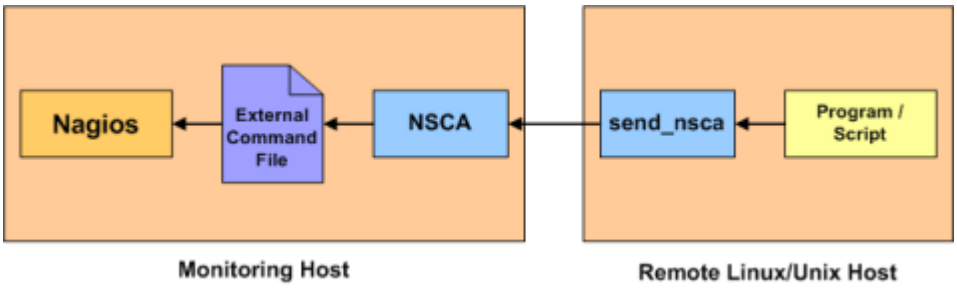
Uses for Passive Checks

Passive checks are useful for monitoring services that are:

- Asynchronous in nature and cannot be monitored effectively by polling their status on a regularly scheduled basis
- Located behind a firewall and cannot be checked actively from the monitoring host

Examples of asynchronous services that lend themselves to being monitored passively include SNMP traps and security alerts. You never know how many (if any) traps or alerts you'll receive in a given time frame, so it's not feasible to just monitor their status every few minutes.

Passive checks are also used when configuring distributed or redundant monitoring installations.



Host Checks

Hosts are checked by the Nagios Core daemon:

- At regular intervals, as defined by the *check_interval* and *retry_interval* options in your host definitions.
- On-demand when a service associated with the host changes state.
- On-demand as needed as part of the host reachability logic.
- On-demand as needed for predictive host dependency checks.

Regularly scheduled host checks are optional. If you set the *check_interval* option in your host definition to zero (0), Nagios Core will not perform checks of the hosts on a regular basis. It will, however, still perform on-demand checks of the host as needed for other parts of the monitoring logic.

On-demand checks are made when a service associated with the host changes state because Nagios Core needs to know whether the host has also changed state. Services that change state are often an indicator that the host may have also changed state. For example, if Nagios Core detects that the HTTP service associated with a host just changed from a CRITICAL to an OK state, it may indicate that the host just recovered from a reboot and is now back up and running.

On-demand checks of hosts are also made as part of the host reachability logic. Nagios Core is designed to detect network outages as quickly as possible, and distinguish between DOWN and UNREACHABLE host states. These are very different states and can help an admin quickly locate the cause of a network outage.

On-demand checks are also performed as part of the predictive host dependency check logic. These checks help ensure that the dependency logic is as accurate as possible.

Cached Host Checks

The performance of on-demand host checks can be significantly improved by implementing the use of cached checks, which allow Nagios Core to forgo executing a host check if it determines a relatively recent check result will do instead.

Dependencies and Checks

You can define host execution dependencies that prevent Nagios Core from checking the status of a host depending on the state of one or more other hosts.

Parallelization of Host Checks

Scheduled host checks are run in parallel. When Nagios Core needs to run a scheduled host check, it will initiate the host check and then return to doing other work (running service checks, etc). The host check runs in a child process that was fork()ed from the main Nagios Core daemon. When the host check has completed, the child process will inform the main Nagios Core process (its parent) of the check results. The main Nagios Core process then handles the check results and takes appropriate action (running event handlers, sending notifications, etc.).

Host States

Hosts that are checked can be in one of three different states:

- UP
- DOWN
- UNREACHABLE

Host State Determination

Host checks are performed by plugins, which can return a state of OK, WARNING, UNKNOWN, or CRITICAL. How does Nagios Core translate these plugin return codes into host states of UP, DOWN, or UNREACHABLE?

The table below shows how plugin return codes correspond with preliminary host states. Some post-processing is done which may then alter the final host state.

Plugin Result	Preliminary Host State
OK	UP
WARNING	UP or DOWN*
UNKNOWN	DOWN
CRITICAL	DOWN

If the preliminary host state is DOWN, Nagios Core will attempt to see if the host is really DOWN or if it is UNREACHABLE. The distinction between DOWN and UNREACHABLE host states is important, as it allows admins to determine root cause of network outages faster. The following table shows how Nagios Core makes a final state determination based on the state of the hosts parent(s). A host's parents are defined in the *parents* directive in host definition.

Preliminary Host State	Parent Host State	Final Host State
DOWN	At least one parent is UP	DOWN
DOWN	All parents are either DOWN or UNREACHABLE	UNREACHABLE

Host State Changes

As you are probably well aware, hosts don't always stay in one state. Things break, patches get applied, and servers need to be rebooted. When Nagios Core checks the status of hosts, it will be able to detect when a host changes between UP, DOWN, and UNREACHABLE states and take appropriate action. These state changes result in different state types (HARD or SOFT), which can trigger event handlers to be run and notifications to be sent out

Host Dependencies

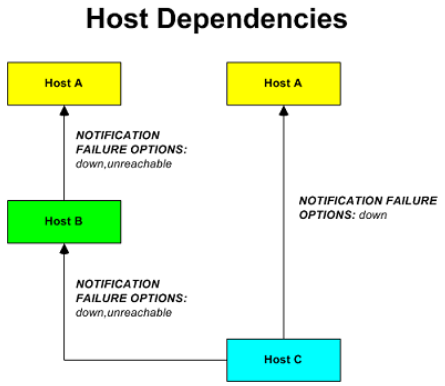
Do not confuse host dependencies with parent/child host relationships. You should be using parent/child host relationships (defined with the *parents* directive in host definitions) for most cases, rather than host dependencies. A description of how parent/child host relationships work can be found in the documentation on network reachability.

Here are the basics about host dependencies:

1. A host can be dependent on one or more other host
2. Host dependencies are not inherited (unless specifically configured to)
3. Host dependencies can be used to cause host check execution and host notifications to be suppressed under different circumstances (UP, DOWN, and/or UNREACHABLE states)
4. Host dependencies might only be valid during specific timeperiods

Example Host Dependencies

The image below shows an example of the logical layout of host notification dependencies. Different hosts are dependent on other hosts for notifications.



In the example above, the dependency definitions for *Host C* would be defined as follows:

```
define hostdependency {
    host_name                       Host A
    dependent_host_name             Host C
    notification_failure_criteria   d
}

define hostdependency {
    host_name                       Host B
    dependent_host_name             Host C
    notification_failure_criteria   d,u
}
```

As with service dependencies, host dependencies are not inherited. In the example image you can see that Host C does not inherit the host dependencies of Host B. In order for Host C to be dependent on Host A, a new host dependency definition must be defined.

Service Checks

Services are checked by the Nagios Core daemon:

- At regular intervals, as defined by the *check_interval* and *retry_interval* options in your service definitions.
- On-demand as needed for predictive service dependency checks.

On-demand checks are performed as part of the predictive service dependency check logic. These checks help ensure that the dependency logic is as accurate as possible. If you don't make use of service dependencies, Nagios Core won't perform any on-demand service checks.

Cached Service Checks

The performance of on-demand service checks can be significantly improved by implementing the use of cached checks, which allow Nagios Core to forgo executing a service check if it determines a relatively recent check result will do instead. Cached checks will only provide a performance increase if you are making use of service dependencies.

Dependencies and Checks

You can define service execution dependencies that prevent Nagios Core from checking the status of a service depending on the state of one or more other services.

Parallelization of Service Checks

Scheduled service checks are run in parallel. When Nagios Core needs to run a scheduled service check, it will initiate the service check and then return to doing other work (running host checks, etc). The service check runs in a child process that was fork()ed from the main Nagios Core daemon. When the service check has completed, the child process will inform the main Nagios Core process (its parent) of the check results. The main Nagios Core process then handles the check results and takes appropriate action (running event handlers, sending notifications, etc.).

Service States

Services that are checked can be in one of four different states:

- OK
- WARNING
- UNKNOWN
- CRITICAL

Service State Determination

Service checks are performed by plugins, which can return a state of OK, WARNING, UNKNOWN, or CRITICAL. These plugin states directly translate to service states. For example, a plugin which returns a WARNING state will cause a service to have a WARNING state.

Services State Changes

When Nagios Core checks the status of services, it will be able to detect when a service changes between OK, WARNING, UNKNOWN, and CRITICAL states and take appropriate action. These state changes result in different state types (HARD or SOFT), which can trigger event handlers to be run and notifications to be sent out. Service state changes can also trigger on-demand host checks. Detecting and dealing with state changes is what Nagios Core is all about.

Macros

One of the main features that make Nagios so flexible is the ability to use macros in command defintions. Macros allow you to reference information from hosts, services, and other sources in your commands.

Macro Substitution - How Macros Work

Before Nagios Core executes a command, it will replace any macros it finds in the command definition with their corresponding values. This macro substitution occurs for all types of commands that Nagios executes - host and service checks, notifications, event handlers, etc.

Certain macros may themselves contain other macros. These include the $HOSTNOTES$, $HOSTNOTESURL$, $HOSTACTIONURL$, $SERVICENOTES$, $SERVICENOTESURL$, and $SERVICEACTIONURL$ macros.

Example 1: Host Address Macro

When you use host and service macros in command definitions, they refer to values for the host or service for which the command is being run. Let's try an example. Assuming we are using a host definition and a *check_ping* command defined like this:

```
define host {
    host_name       linuxbox
    address         192.168.50.2
    check_command   check_ping
    ...
}

define command {
    command_name    check_ping
    command_line    /usr/local/nagios/libexec/check_ping -H $HOSTADDRESS$ -w 100.0,90%
  -c 200.0,60%
}
```

The expanded/final command line to be executed for the host's check command would look like this:

```
/usr/local/nagios/libexec/check_ping -H 192.168.50.2 -w 100.0,60% -c 200.0,90%
```

Configurations:

nagios.cfg file:

This is the main configuration file of Nagios core. This file contains the location of log file of Nagios, hosts and services state update interval, lock file and status.dat file. Nagios users and groups on which the instances are running are defined in this file. It has path of all the individual object config files like commands, contacts, templates etc.

Example:

```
# LOG FILE
# This is the main log file where service and host events are logged
# for historical purposes.  This should be the first option specified
# in the config file!!!

log_file=/usr/local/nagios/var/nagios.log

# OBJECT CONFIGURATION FILE(S)
# These are the object configuration files in which you define hosts,
# host groups, contacts, contact groups, services, etc.
# You can split your object definitions across several config files
# if you wish (as shown below), or keep them all in a single config file.

# You can specify individual object config files as shown below:

cfg_file=/usr/local/nagios/etc/objects/commands.cfg
cfg_file=/usr/local/nagios/etc/objects/contacts.cfg
cfg_file=/usr/local/nagios/etc/objects/timeperiods.cfg
cfg_file=/usr/local/nagios/etc/objects/templates.cfg

cfg_file=/usr/local/nagios/etc/objects/router.cfg
cfg_file=/usr/local/nagios/etc/objects/routers.cfg

# Definitions for monitoring the local (Linux) host
cfg_file=/usr/local/nagios/etc/objects/linux.cfg
```

cgi.cfg file:

By default, the CGI configuration file of Nagios is named cgi.cfg. It tells the CGIs where to find the main configuration file. The CGIs will read the main and host config files for any other data they might need. It contains all the user and group information and their rights and permissions. It also has the path for all frontend files of Nagios.

Example:

```
##############################################################
#
# CGI.CFG - Sample CGI Configuration File for Nagios 4.4.6
#
##############################################################
```

```
# MAIN CONFIGURATION FILE
# This tells the CGIs where to find your main configuration file.
# The CGIs will read the main and host config files for any other
# data they might need.

main_config_file=/usr/local/nagios/etc/nagios.cfg

# PHYSICAL HTML PATH
# This is the path where the HTML files for Nagios reside.  This
# value is used to locate the logo images needed by the statusmap
# URL HTML PATH
# This is the path portion of the URL that corresponds to the
# physical location of the Nagios HTML files (as defined above).
# This value is used by the CGIs to locate the online documentation
# and graphics.  If you access the Nagios pages with an URL like
# http://www.myhost.com/nagios, this value should be '/nagios'
# (without the quotes).

url_html_path=/nagios
```

resource.cfg File:

You can define $USERx$ macros in this file, which can in turn be used in command definitions in your host config file(s). $USERx$ macros are useful for storing sensitive information such as usernames, passwords, etc.

They are also handy for specifying the path to plugins and event handlers - if you decide to move the plugins or event handlers to a different directory in the future, you can just update one or two $USERx$ macros, instead of modifying a lot of command definitions. Resource files may also be used to store configuration directives for external data sources like MySQL

```
# Sets $USER1$ to be the path to the plugins
$USER1$=/usr/local/nagios/libexec

# Sets $USER2$ to be the path to event handlers
# $USER2$=/usr/local/nagios/libexec/eventhandlers
```

commands.cfg

This config file provides you with some example command definitions that you can refer in host, service, and contact definitions. These commands are used to check and monitor hosts and services. You can run these commands locally on a Linux console where you will also get the output of the command you run.

Example:

```
define command {
   command_name check_local_disk
   command_line $USER1$/check_disk -w $ARG1$ -c $ARG2$ -p $ARG3$
}

define command {
   command_name check_local_load
```

```
    command_line $USER1$/check_load -w $ARG1$ -c $ARG2$
}

define command {
    command_name check_local_procs
    command_line $USER1$/check_procs -w $ARG1$ -c $ARG2$ -s $ARG3$
}
```

contacts.cfg:

This file contains contacts and groups information of Nagios. By default, one contact is already present Nagios admin.

Example:

```
###############################################################################
# CONTACTS
###############################################################################
# Just one contact defined by default - the Nagios admin (that's you)
# This contact definition inherits a lot of default values from the
# 'generic-contact' template which is defined elsewhere.

define contact {
    contact_name nagiosadmin
    use generic-contact
    alias Nagios Admin
    email mahedi@bitbyte.net.bd
}

###############################################################################
# CONTACT GROUPS
###############################################################################
# We only have one contact in this simple configuration file, so there is
# no need to create more than one contact group.

define contactgroup {
    contactgroup_name admins
    alias Nagios Administrators
    members nagiosadmin
}
```

templates.cfg file:

This config file provides you with some example object definition templates that are referred by other host, service, contact, etc. definitions in other config files.

```
###############################################################################
#
# CONTACT TEMPLATES
#
###############################################################################

# Generic contact definition template
# This is NOT a real contact, just a template!

define contact {

    name                            generic-contact         ; The name of this contact template
    service_notification_period     24x7                    ; service notifications can be sent anytime
    host_notification_period        24x7                    ; host notifications can be sent anytime
    service_notification_options    w,u,c,r,f,s             ; send notifications for all service states, flapping events, and scheduled downtime events
    host_notification_options       d,u,r,f,s               ; send notifications for all host states, flapping events, and scheduled downtime events
```

```
    service_notification_commands   notify-service-by-email ; send service notifications via email
    host_notification_commands      notify-host-by-email    ; send host notifications via email
    register                        0                       ; DON'T REGISTER THIS DEFINITION - ITS NOT A REAL CONTACT, JUST A TEMPLATE!
}
###############################################################################
#
# HOST TEMPLATES
#
###############################################################################

# Generic host definition template
# This is NOT a real host, just a template!

define host {

    name                            generic-host            ; The name of this host template
    notifications_enabled           1                       ; Host notifications are enabled
    event_handler_enabled           1                       ; Host event handler is enabled
    flap_detection_enabled          1                       ; Flap detection is enabled
    process_perf_data               1                       ; Process performance data
    retain_status_information       1                       ; Retain status information across program restarts
    retain_nonstatus_information    1                       ; Retain non-status information across program restarts
    notification_period             24x7                    ; Send host notifications at any time
    register                        0                       ; DON'T REGISTER THIS DEFINITION - ITS NOT A REAL HOST, JUST A TEMPLATE!
}
# Linux host definition template
# This is NOT a real host, just a template!

define host {

    name                            linux-server            ; The name of this host template
    use                             generic-host            ; This template inherits other values from the generic-host template
    check_period                    24x7                    ; By default, Linux hosts are checked round
the clock
    check_interval                  5                       ; Actively check the host every 5 minutes
    retry_interval                  1                       ; Schedule host check retries at 1 minute intervals
    max_check_attempts              10                      ; Check each Linux host 10 times (max)
    check_command                   check-host-alive        ; Default command to check Linux hosts
    notification_period             workhours               ; Linux admins hate to be woken up, so we only notify during the day
                                                            ; Note that the notification_period variable is being overridden from
                                                            ; the value that is inherited from the generic-host template!
    notification_interval           120                     ; Resend notifications every 2 hours
    notification_options            d,u,r                   ; Only send notifications for specific host
states
    contact_groups                  admins                  ; Notifications get sent to the admins by default
   register                        0                       ; DON'T REGISTER THIS DEFINITION - ITS NOT A REAL HOST, JUST A TEMPLATE!
}
# Windows host definition template
# This is NOT a real host, just a template!

define host {

    name                            windows-server          ; The name of this host template
    use                             generic-host            ; Inherit default values from the generic-host template
    check_period                    24x7                    ; By default, Windows servers are monitored
round the clock
    check_interval                  5                       ; Actively check the server every 5 minutes
    retry_interval                  1                       ; Schedule host check retries at 1 minute intervals
    max_check_attempts              10                      ; Check each server 10 times (max)
    check_command                   check-host-alive        ; Default command to check if servers are "alive"
    notification_period             24x7                    ; Send notification out at any time - day or night
    notification_interval           30                      ; Resend notifications every 30 minutes
    notification_options            d,r                     ; Only send notifications for specific host
states
    contact_groups                  admins                  ; Notifications get sent to the admins by default
    hostgroups                      windows-servers         ; Host groups that Windows servers should be a member of
a member of
    register                        0                       ; DON'T REGISTER THIS - ITS JUST A TEMPLATE
}
```

timeperiods.cfg file:

This config file provides you with some example time period definitions that you can refer in host, service, contact, and dependency definitions.

```
###############################################################################
# TIMEPERIOD DEFINITIONS
###############################################################################

# This defines a timeperiod where all times are valid for checks,
# notifications, etc.  The classic "24x7" support nightmare. :-)

define timeperiod {
```

```
    name                    24x7
    timeperiod_name         24x7
    alias                   24 Hours A Day, 7 Days A Week

    sunday                  00:00-24:00
    monday                  00:00-24:00
    tuesday                 00:00-24:00
    wednesday               00:00-24:00
    thursday                00:00-24:00
    friday                  00:00-24:00
    saturday                00:00-24:00
}

# This defines a timeperiod that is normal workhours for
# those of us monitoring networks and such in the U.S.

define timeperiod {

    name                    workhours
    timeperiod_name         workhours
    alias                   Normal Work Hours

    monday                  09:00-17:00
    tuesday                 09:00-17:00
    wednesday               09:00-17:00
    thursday                09:00-17:00
    friday                  09:00-17:00
}
```

Monitor Remote Linux Host Using NRPE and Nagios Plugins:

Install NRPE:

```
# dnf install epel-release

# dnf install nrpe
```

Confirm installed version once the installation completes.

```
nrpe -V
```

NRPE - Nagios Remote Plugin Executor

```
Version: 3.2.1
```

Running NRPE

Run the commands below to start and enable nrpe to run on system boot.

```
systemctl enable nrpe --now
```

To check the status of NRPE agent;

```
systemctl status nrpe
```

Open NRPE Port on Firewalld

NRPE uses port TCP 5666 by default. If firewallD is running, open this port to allow external access esp from Nagios Monitoring server.

```
firewall-cmd --add-port=5666/tcp --permanent
firewall-cmd --reload
```

Configure NRPE Add-on

Modify the NRPE configuration file to accept the connection from the Nagios server, Edit the /etc/nagios/nrpe.cfg file.

```
vi /etc/nagios/nrpe.cfg

allowed_hosts=192.168.20.120
```

Install Nagios Plugins on CentOS 8

Install Required Build Tools and Dependencies

```
 dnf install gcc glibc glibc-common make gettext automake autoconf wget openssl-devel
 @perl wget unzip glibc automake glibc-common gettext autoconf gcc gd gd-devel
 openssl-devel net-snmp postfix net-snmp-utils
```

```
# dnf install epel-release

# dnf config-manager --set-enabled powertools

# dnf install nagios-plugins-all
```

Test Nagios Checks

For example, execute the below command in another terminal to see the check result.

```
# /usr/lib64/nagios/plugins/check_procs -w 200 -c 250
PROCS OK: 87 processes | procs=87;200;250;0;
```

Check the following commands in `/etc/nagios/nrpe.cfg` file

```
command[check_users]=/usr/lib64/nagios/plugins/check_users -w 5 -c 10
command[check_load]=/usr/lib64/nagios/plugins/check_load -r -w .15,.10,.05 -c .30,.25,.20
command[check_hda1]=/usr/lib64/nagios/plugins/check_disk -w 20% -c 10% -p /dev/hda1
command[check_zombie_procs]=/usr/lib64/nagios/plugins/check_procs -w 5 -c 10 -s Z
command[check_total_procs]=/usr/lib64/nagios/plugins/check_procs -w 150 -c 200
command[check_root]=/usr/lib64/nagios/plugins/check_disk -w 20% -c 10% -p /
command[check_swap]=/usr/lib64/nagios/plugins/check_swap -w 20% -c 10%
```

Restart NRPE:

```
# systemctl restart nrpe
```

On Nagios Server

Install NRPE plugin

This NRPE plugin provides check_nrpe plugin which contacts the NRPE server on remote machines to check the services or resource.

CentOS / RHEL

Nagios NRPE plugin is available in the EPEL repository for CentOS / RHEL. So, configure the EPEL repository your CentOS / RHEL system

Use the following command to install the NRPE plugin on your machine.

```
 yum install epel-release
 yum -y install nagios-plugins-nrpe
```

Add Command Definition

Now it's time to configure the Nagios server to monitor the remote client machine, and You'll need to create a command definition in Nagios object configuration file to use the check_nrpe plugin.

Open the commands.cfg file.

```
vi /usr/local/nagios/etc/objects/commands.cfg
```

Add the following Nagios command definition to the file.

#check_nrpe. command definition

```
define command {
command_name check_nrpe
command_line /usr/lib64/nagios/plugins/check_nrpe -H $HOSTADDRESS$ -t 30 -c $ARG1$
}
```

Add a Linux host to Nagios server

Create a client configuration file /usr/local/nagios/etc/servers/centos8-1.cfg to define the host and service definitions of remote Linux host.

```
vi /usr/local/nagios/etc/servers/centos8-1.cfg
```

```
define host{

            use                     linux-server
            host_name               centos8-1
            alias                   centos8-1
            hostgroups              linux-servers
            address                 192.168.90.2

}
```

```
define service{

            use                     local-service
            host_name               centos-1
            service_description     SWAP Uasge
            check_command           check_nrpe!check_swap

}

define service{

            use                     local-service
            host_name               centos-1
            service_description     Root / Partition
            check_command           check_nrpe!check_root

}

define service{

            use                     local-service
            host_name               centos8-1
            service_description     Current Users
            check_command           check_nrpe!check_users

}

define service{

            use                     local-service
            host_name               centos-1
            service_description     Total Processes
            check_command           check_nrpe!check_total_procs

}

define service{

            use                     local-service
            host_name               centos-1
            service_description     Current Load
            check_command           check_nrpe!check_load

}
```

Verify Nagios for any errors:

```
/usr/local/nagios/bin/nagios -v /usr/local/nagios/etc/nagios.cfg
```

Restart the Nagios server:

```
systemctl restart nagios
```

Configure Notification:

Install mail clients:

```
# dnf install mailx
```

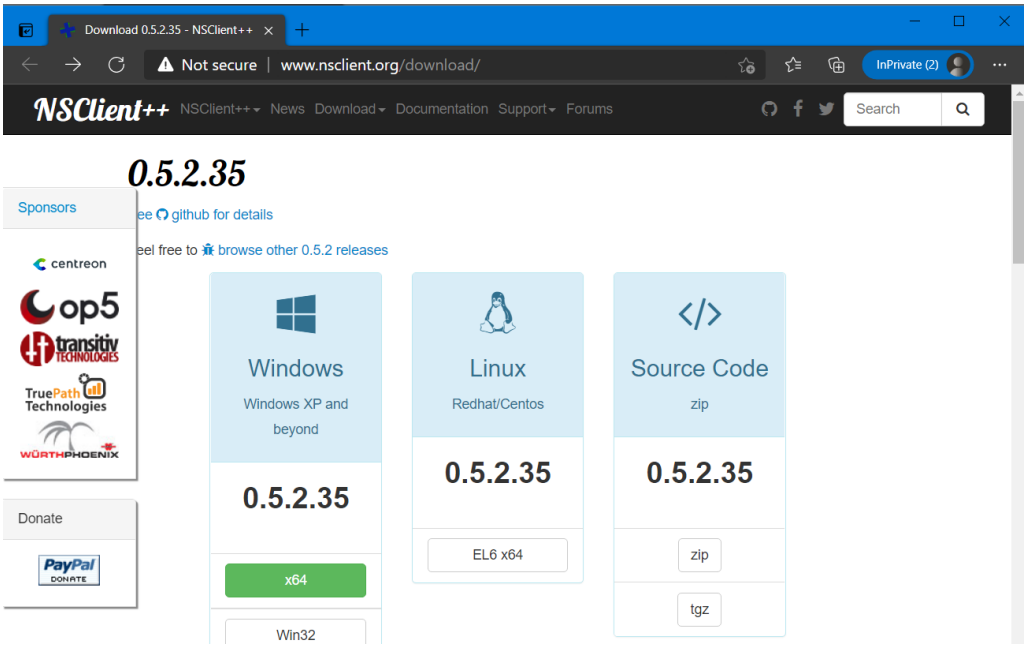
Update command in /usr/local/nagios/etc/objects/commands.cfg file

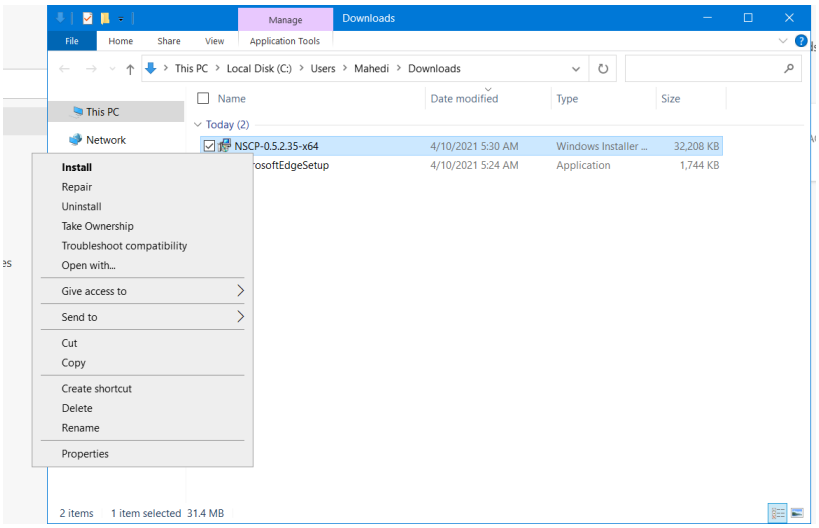
```
define command {
  command_name    notify-host-by-email
  command_line    /usr/bin/printf "%b" "***** Nagios *****\n\nNotification Type:
 $NOTIFICATIONTYPE$\nHost: $HOSTNAME$\nState: $HOSTSTATE$\nAddress:
 $HOSTADDRESS$\nInfo: $HOSTOUTPUT$\n\nDate/Time: $LONGDATETIME$\n" | /bin/mail -s "**
 $NOTIFICATIONTYPE$ Host Alert: $HOSTNAME$ is $HOSTSTATE$ **" $CONTACTEMAIL$
}


define command {

    command_name    notify-service-by-email
    command_line    /usr/bin/printf "%b" "***** Nagios *****\n\nNotification Type:
 $NOTIFICATIONTYPE$\n\nService: $SERVICEDESC$\nHost: $HOSTALIAS$\nAddress:
 $HOSTADDRESS$\nState: $SERVICESTATE$\n\nDate/Time: $LONGDATETIME$\n\nAdditional
 Info:\n\n$SERVICEOUTPUT$\n" | /bin/mail -s "** $NOTIFICATIONTYPE$ Service Alert:
 $HOSTALIAS$/$SERVICEDESC$ is $SERVICESTATE$ **" $CONTACTEMAIL$
}
```

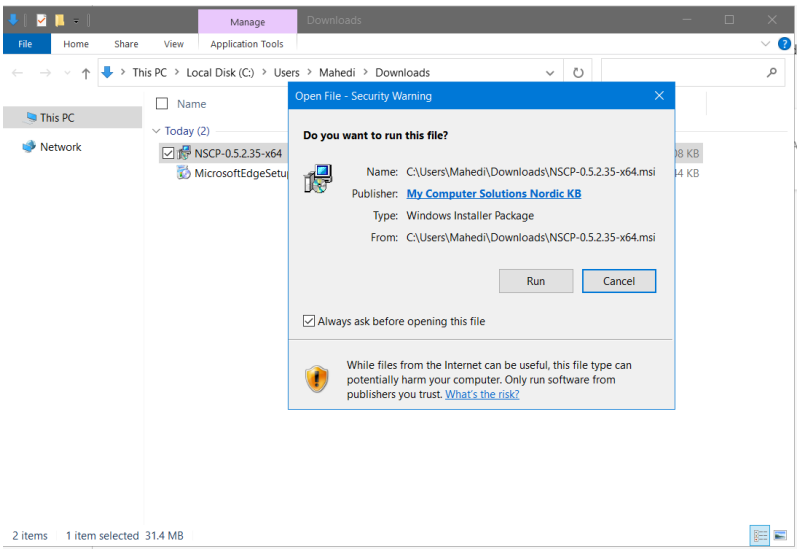
Update email address in in /usr/local/nagios/etc/objects/contacts.cfg file

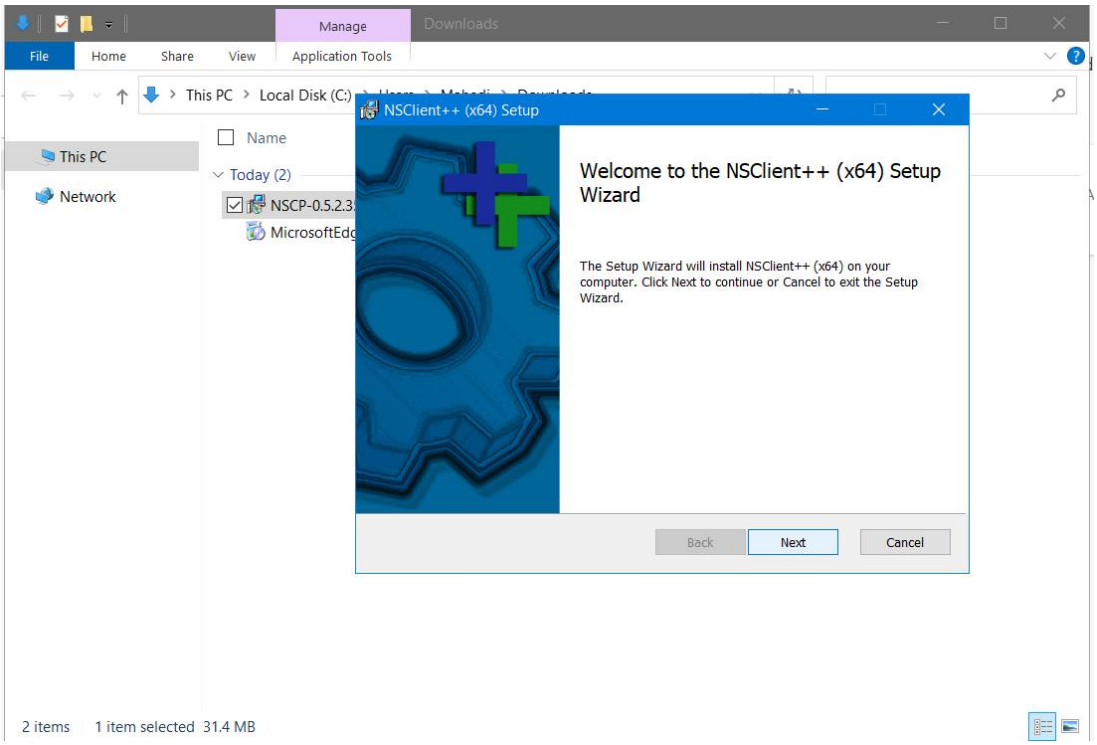
```
define contact {

    contact_name            nagiosadmin             ; Short name of user
    use                     generic-contact         ; Inherit default values from generic-contact
  template (defined above)
    alias                   Nagios Admin            ; Full name of user
    email                   mahedi@bitbyte.net.bd ; <<***** CHANGE THIS TO YOUR EMAIL ADDRESS *****
}
```

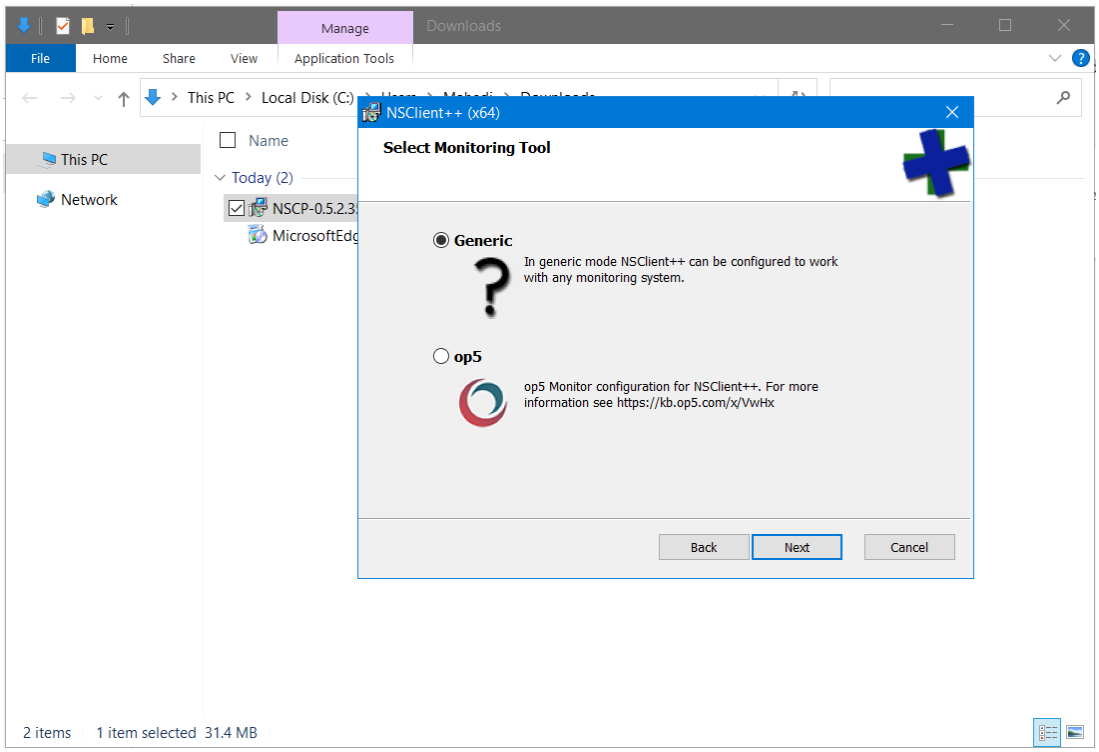
Monitor Windows Host:

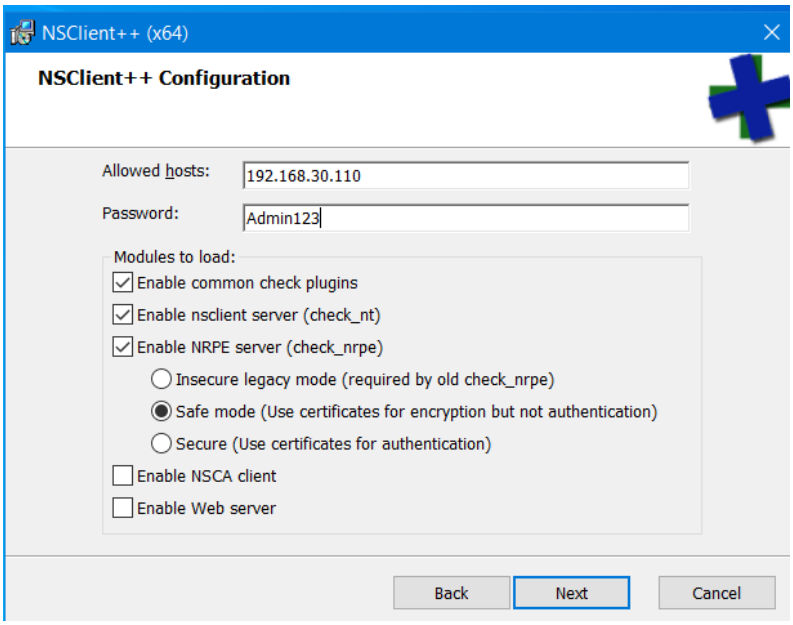

Download 0.5.2.35 - NSClient++
Not secure | www.nsclient.org/download/
InPrivate (2)
NSClient++
NSClient++ News Download Documentation Support Forums
Search
0.5.2.35
Sponsors
ee github for details
eel free to browse other 0.5.2 releases
centreon
op5
transitivity TECHNOLOGIES
TruePath Technologies
WÜRTHPHOENIX
Donate
PayPal DONATE
Windows
Windows XP and beyond
0.5.2.35
x64
Win32
Linux
Redhat/Centos
0.5.2.35
EL6 x64
Source Code
zip
0.5.2.35
zip
tgz

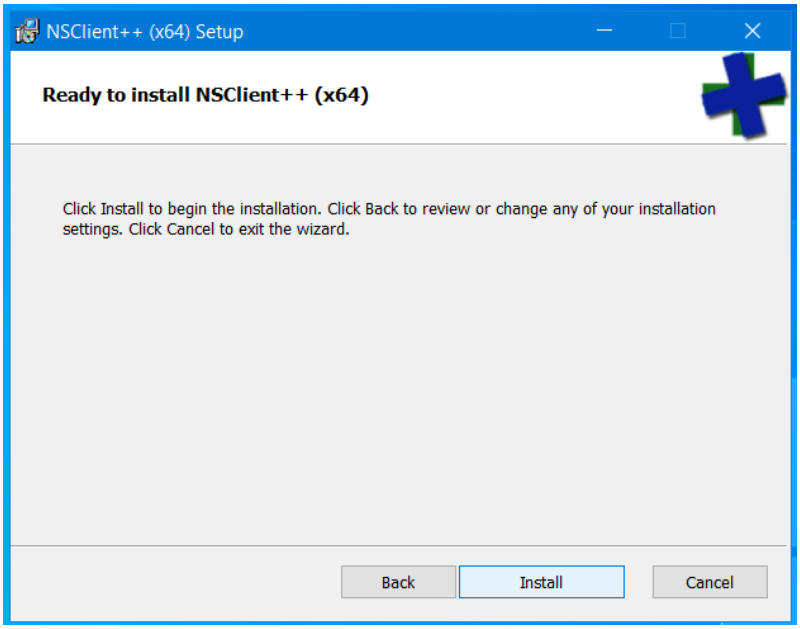

Manage
Downloads
File
Home
Share
View
Application Tools
This PC > Local Disk (C:) > Users > Mahedi > Downloads
This PC
Network
Name
Date modified
Type
Size
Today (2)
NSCP-0.5.2.35-x64
4/10/2021 5:30 AM
Windows Installer ...
32,208 KB
MicrosoftEdgeSetup
4/10/2021 5:24 AM
Application
1,744 KB
Install
Repair
Uninstall
Take Ownership
Troubleshoot compatibility
Open with...
Give access to
Send to
Cut
Copy
Create shortcut
Delete
Rename
Properties
2 items
1 item selected 31.4 MB

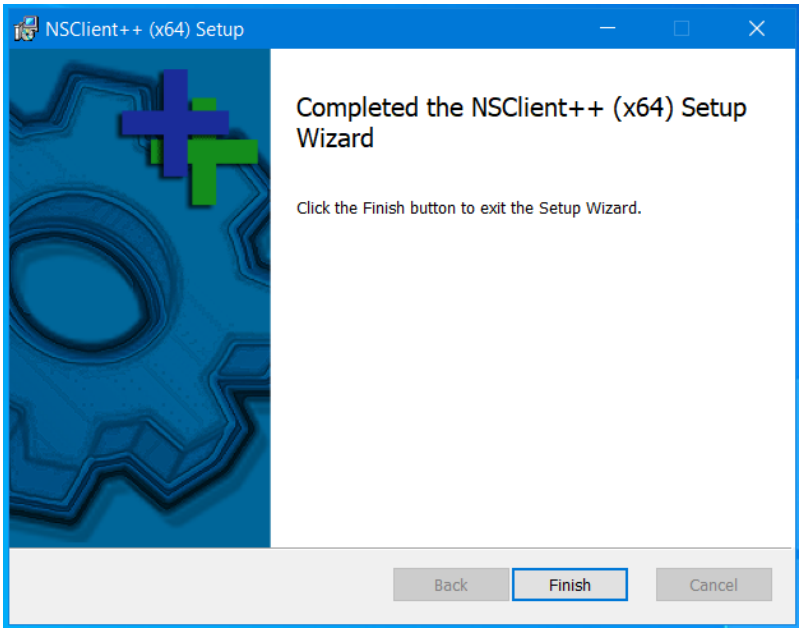

Manage
Downloads
File
Home
Share
View
Application Tools
This PC > Local Disk (C:) > Users > Mahedi > Downloads
This PC
Network
Name
Today (2)
NSCP-0.5.2.35-x64
Open File - Security Warning
Do you want to run this file?
Name: C:\Users\Mahedi\Downloads\NSCP-0.5.2.35-x64.msi
Publisher: My Computer Solutions Nordic KB
Type: Windows Installer Package
From: C:\Users\Mahedi\Downloads\NSCP-0.5.2.35-x64.msi
Run
Cancel
Always ask before opening this file
While files from the Internet can be useful, this file type can potentially harm your computer. Only run software from publishers you trust. What's the risk?
2 items
1 item selected 31.4 MB

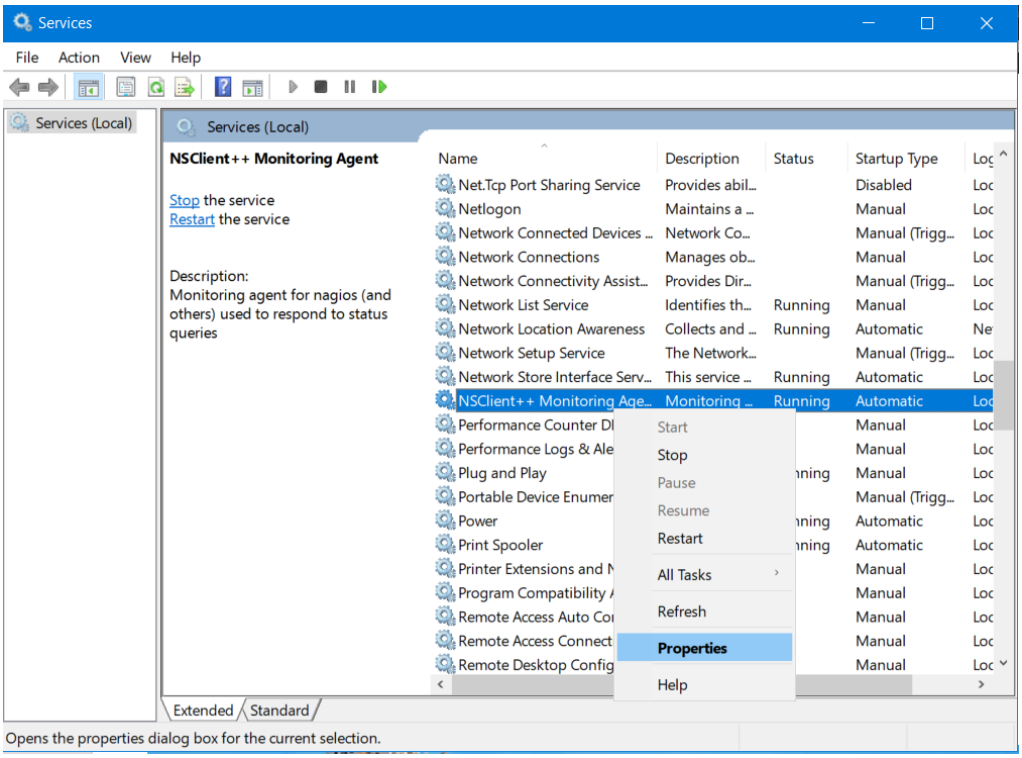

Manage
Downloads
File
Home
Share
View
Application Tools
This PC
Local Disk (C:)
Name
This PC
Network
Today (2)
NSCP-0.5.2.3
MicrosoftEdg
2 items
1 item selected 31.4 MB
NSClient++ (x64) Setup
Welcome to the NSClient++ (x64) Setup Wizard
The Setup Wizard will install NSClient++ (x64) on your computer. Click Next to continue or Cancel to exit the Setup Wizard.
Back
Next
Cancel

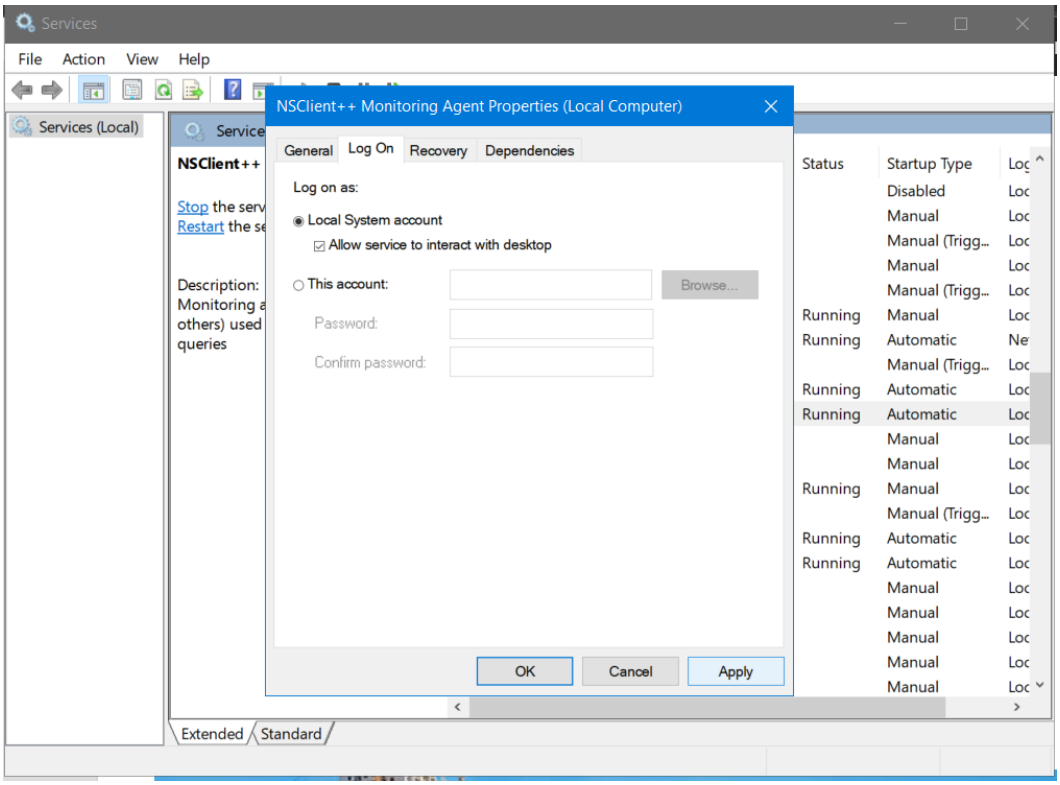

Manage
Downloads
File
Home
Share
View
Application Tools
This PC
Local Disk (C:)
Name
This PC
Network
Today (2)
NSCP-0.5.2.3
MicrosoftEdg
2 items
1 item selected 31.4 MB
NSClient++ (x64)
Select Monitoring Tool
Generic
In generic mode NSClient++ can be configured to work with any monitoring system.
op5
op5 Monitor configuration for NSClient++. For more information see https://kb.op5.com/x/VwHx
Back
Next
Cancel


NSClient++ (x64)
NSClient++ Configuration
Allowed hosts:
192.168.30.110
Password:
Admin123
Modules to load:
Enable common check plugins
Enable nsclient server (check_nt)
Enable NRPE server (check_nrpe)
Insecure legacy mode (required by old check_nrpe)
Safe mode (Use certificates for encryption but not authentication)
Secure (Use certificates for authentication)
Enable NSCA client
Enable Web server
Back
Next
Cancel


NSClient++ (x64) Setup
Ready to install NSClient++ (x64)
Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.
Back
Install
Cancel


NSClient++ (x64) Setup
Completed the NSClient++ (x64) Setup Wizard
Click the Finish button to exit the Setup Wizard.
Back
Finish
Cancel


Services
File Action View Help
Services (Local)
NSClient++ Monitoring Agent
Stop the service
Restart the service
Description:
Monitoring agent for nagios (and others) used to respond to status queries
Name
Description
Status
Startup Type
Net.Tcp Port Sharing Service
Netlogon
Network Connected Devices ...
Network Connections
Network Connectivity Assist...
Network List Service
Network Location Awareness
Network Setup Service
Network Store Interface Serv...
NSClient++ Monitoring Age...
Performance Counter DL
Performance Logs & Ale
Plug and Play
Portable Device Enumer
Power
Print Spooler
Printer Extensions and N
Program Compatibility A
Remote Access Auto Co
Remote Access Connect
Remote Desktop Config
Start
Stop
Pause
Resume
Restart
All Tasks
Refresh
Properties
Help
Extended
Standard
Opens the properties dialog box for the current selection.


Services
File Action View Help
Services (Local)
NSClient++ Monitoring Agent Properties (Local Computer)
General
Log On
Recovery
Dependencies
Log on as:
Local System account
Allow service to interact with desktop
This account:
Browse...
Password:
Confirm password:
OK
Cancel
Apply
Extended
Standard